CONTEXT SWITCHED ROUTE LOOKUP KEY ENGINE

BACKGROUND OF THE INVENTION

Field of the Invention

[0001] The present invention relates generally to data processing and, more particularly, to systems and methods for performing route lookups for packets of information.

Description of Related Art

[0002] Routers receive data on a physical media, such as optical fiber, analyze the data to determine its destination, and output the data on a physical media in accordance with the destination. Routers were initially designed using a general purpose processor executing large software programs. As line rates and traffic volume increased, however, general purpose processors could not scale to meet these new demands. For example, as functionality was added to the software, such as accounting and policing functionality, these routers suffered performance degradation. In some instances, the routers failed to handle traffic at line rate when the new functionality was turned on.

[0003] To meet the new demands, purpose-built routers were designed. Purpose-built routers were planned and constructed with components optimized for routing. They not only handled higher line rates and higher network traffic volume, they also added functionality without compromising line rate performance.

[0004] A purpose-built router may include a number of input and output ports from which it transmits and receives information packets. A switching fabric or other transmission medium may be implemented in the router to carry the packets between the ports. In a high-performance purpose-built router, the switching fabric may transmit a large amount of information between a

number of internal components. Typically, the information is transmitted within the router in discrete quantities, or "cells," which it generates by breaking down information packets that it receives.

[0005] These cells may be routed through the switching fabric or to certain output ports based on a route lookup that is performed by a routing unit. Although the routing units in the first purpose-built routers met the demands of the network at that time, they will not be able to meet the rising demands for bandwidth and added functionality as line rates and network traffic volume increase.

[0006] Thus, there is a need in the art to more efficiently implement route lookups within routers.

SUMMARY OF THE INVENTION

[0007] Systems and methods consistent with the principles of the invention, among other things, process multiple keys per key engine and fully utilize processing circuitry therein by context-switching keys for processing, instead of idly waiting for data and/or instructions to return from a memory.

[0008] In accordance with one purpose of the invention as embodied and broadly described herein, a method of performing route lookups for a group of data may include processing, by a processor, a first data to generate routing information until first information is needed, and requesting the first information. First context state information for the first data may be stored, and the processor may process a second data to generate routing information until second information is needed. The second information may be requested, and second context state

information for the second data may be stored. Processing may resume on the first data using the stored first context state information after the requested first information is received.

[0009] In another implementation consistent with principles of the invention, a method of processing for routing packets may include processing a first data related to routing of a first packet until first information is needed, and requesting the first information. Intermediate information related to the first data may be stored, and a second data related to routing of a second packet may be processed while waiting for the requested first information to arrive.

[0010] In still another implementation consistent with principles of the invention, a method for routing packets of information using corresponding data structures may include receiving a group of data structures related to the packets of information, and sending the data structures to processing engines. Each data structure may correspond to a different packet of information. Each key processor may concurrently perform route lookups for at least two of the data structures at a time. The data structures may be modified based on the route lookups, and the packets of information may be routed based on the modified data structures.

[0011] In further implementation consistent with principles of the invention, a network device may include an input portion configured to receive data structures and to transmit data items associated with the data structures, and a group of processing engines. Each processing engine may be configured to receive a group of data items from the input portion and to contemporaneously compute routes for the data items. A resource may be configured to receive requests from the processing engines. A result processor may be configured to modify the data structures based on the routes computed by the processing engines.

[0012] In yet another implementation consistent with principles of the invention, a system for performing route lookups for a group of data items may include a data processing portion

configured to process one data item at a time and to request data when needed. A buffer may be configured to store a partial result from the data processing portion. A controller may be configured to load the partial result from the data processing portion into the buffer. The controller also may be configured to input another data item into the data processing portion for processing while requested data is obtained for a prior data item.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate an embodiment of the invention and, together with the description, explain the invention. In the drawings,

[0014] Fig. 1 is a diagram of an exemplary network device in which systems and methods consistent with the principles of invention may be implemented;

[0015] Fig. 2 is an exemplary diagram of a packet forwarding engine (PFE) of Fig. 1 according to an implementation consistent with the principles of invention;

[0016] Fig. 3 is a detailed block diagram illustrating portions of the routing unit shown in Fig. 2 according to an implementation consistent with the principles of invention;

[0017] Fig. 4 is a detailed block diagram illustrating portions of the key engines shown in Fig. 3 according to an implementation consistent with the principles of invention;

[0018] Fig. 5 is an exemplary timing diagram illustrating the context switching performed by the key engine of Fig. 4 according to an implementation consistent with the principles of invention;

[0019] Figs. 6 and 7 are flowcharts of exemplary processing of a packet by the network device of Fig. 1 according to an implementation consistent with the principles of invention;

[0020] Fig. 8 is a flow chart illustrating processing performed by the routing unit in Fig. 3 according to an implementation consistent with the principles of the invention; and

[0021] Fig. 9 is a flow chart illustrating processing performed by the key engine in Fig. 4 according to an implementation consistent with the principles of the invention.

DETAILED DESCRIPTION

[0022] The following detailed description of the invention refers to the accompanying drawings. The same reference numbers may be used in different drawings to identify the same or similar elements. Also, the following detailed description does not limit the invention. Instead, the scope of the invention is defined by the appended claims and equivalents.

[0023] As described herein, in one implementation, a key engine may concurrently process multiple keys by saving a processing state in a buffer and loading another for processing when data and/or instructions are requested from a memory. Double data rate memory may also be used to reduce latency.

SYSTEM DESCRIPTION

[0024] Fig. 1 is a diagram of an exemplary network device in which systems and methods consistent with the principles of the invention may be implemented. The principles of the invention will be described in terms of packets, but the principles apply to flow of any type of data unit. In this particular implementation, the network device takes the form of a router 100. The router 100 may receive one or more data streams from a physical link, process the data stream(s) to determine destination information, and transmit the data stream(s) on one or more links in accordance with the destination information.

[0025] Router 100 may include a routing engine (RE) 110 and multiple packet forwarding engines (PFEs) 120 interconnected via a switch fabric 130. Switch fabric 130 may include one or more switching planes to facilitate communication between two or more of PFEs 120. In an implementation consistent with the principles of the invention, each of the switching planes includes a three-stage switch of crossbar elements.

[0026] RE 110 may include processing logic that performs high level management functions for router 100. For example, RE 110 may communicate with other networks and systems connected to router 100 to exchange information regarding network topology. RE 110 may create routing tables based on the network topology information, create forwarding tables based on the routing tables, and forward the forwarding tables to PFEs 120. PFEs 120 may use the routing tables to perform route lookup for incoming packets. RE 110 may also perform other general control and monitoring functions for router 100.

[0027] Each of PFEs 120 connects to RE 110 and switch fabric 130. PFEs 120 receive data on physical links connected to a network, such as a wide area network (WAN). Each physical link could be one of many types of transport media, such as optical fiber or Ethernet cable. The data on the physical link is formatted according to one of several protocols, such as the synchronous optical network (SONET) standard, an asynchronous transfer mode (ATM) technology, or Ethernet.

[0028] Fig. 2 is an exemplary diagram of a PFE 120 according to an implementation consistent with the present invention. PFE 120 may include physical interface cards (PICs) 210 and 220 connected to a flexible port concentrator (FPC) 230. While two PICs 210 and 220 are shown in Fig. 2, there may be more or fewer PICs in other implementations consistent with the principles of the invention.

[0029] PICs 210 and 220 connect to WAN physical links and FPC 230 and transport data between the WAN and FPC 230. Each of PICs 210 and 220 includes interfacing, processing, and memory elements necessary to transmit data between a WAN physical link and FPC 230. Each of PICs 210 and 220 may be designed to handle a particular type of physical link. For example, a particular PIC may be provided to handle only Ethernet communications.

[0030] For incoming data, PICs 210 and 220 may strip off the layer 1 (L1) protocol information and forward the remaining data (raw packets) to FPC 230. For outgoing data, the PICs 210 and 220 may receive packets from FPC 230, encapsulate the packets in L1 protocol information, and transmit the data on the physical WAN link.

[0031] FPC 230 performs packet transfers between PICs 210 and 220 and switch fabric 130. For each packet it handles, FPC 230 may perform route lookup based on packet header information to determine destination information and send the packet either to PIC 210 and 220 or switch fabric 130, depending on the destination information.

[0032] FPC 230 may include processing units 232 and 234, first input/output (I/O) logic 236, second I/O logic 238, memory system 240, and a routing (R) unit 242. Each of processing units 232 and 234 corresponds to one of PICs 210 and 220. Processing units 232 and 234 may process packet data flowing between PICs 210 and 220, respectively, and first I/O logic 236. Each of processing units 232 and 234 may operate in two modes: a first mode for processing packet data received from PIC 210 or 220 connected to it, and a second mode for processing packet data received from first I/O logic 236.

[0033] In the first mode, processing unit 232 or 234 may process packets from PIC 210 or 220, respectively, convert the packets into "cells," and transmit the cells to first I/O logic 236.

Cells are the data structure used internally by FPC 230 for transporting and storing data. In one implementation, cells are 64 bytes in length.

[0034] Packets received by processing unit 232 or 234 may include two portions: a header portion and a packet data portion. For each packet, processing unit 232 or 234 may process the header and insert the header and processing results into the cells. For example, processing unit 232 or 234 may parse layer 2 (L2) and layer 3 (L3) headers of incoming packets. Processing unit 232 or 234 may also create control information based on the packet. The control information may be based on the packet header, the packet data, or both. Processing unit 232 or 234 may then store the parsed headers, control information, and the packet data in cells, which it sends to first I/O logic 236.

[0035] In the second mode, processing unit 232 or 234 handles data flow in the opposite direction to the first mode. In the second mode, processing unit 232 or 234 receives cells from first I/O logic 236, extracts the header information, control information, and packet data from the cells, and creates a packet based on the extracted information. Processing unit 232 or 234 creates the packet header from the header information and possibly the control information from the cells. In one implementation, processing unit 232 or 234 creates L2 and L3 header information based on the header information and control information. Processing unit 232 or 234 may load the packet data portion with the packet data from the cells.

[0036] First I/O logic 236 and second I/O logic 238 coordinate data transfers into and out of FPC 230. First I/O logic 236 and second I/O logic 238 also create data structures called "notifications" based on L2/L3 header information and control information in the cells. While first I/O logic 236 and second I/O logic 238 are shown as separate units, they may be implemented as a single unit in other implementations consistent with principles of the invention.

[0037] Memory system 240 may temporarily store cells from first I/O logic 236 and second I/O logic 238, as well as notifications from R unit 242.

[0038] R unit 242 receives notifications from first I/O logic 236 and second I/O logic 238. R unit 242 may include processing logic that provides route lookup, accounting, and policing functionality. R unit 242 may receive one or more routing tables from RE 110 (Fig. 1) and use the routing table(s) to perform route lookups based on the notifications. R unit 242 may insert the lookup result into the notification, which it forwards to memory system 240.

R UNIT DESCRIPTION

[0039] Fig. 3 shows an embodiment of R unit 242 consistent with the principles of the invention. R unit 242 provides route lookup, encapsulation lookup, and filtering for cells coming from first I/O logic 236 and second I/O logic 238. For an incoming packet from either I/O logic 236/238, R unit 242 receives a notification, which includes a "key" that contains L2/L3 header information. R unit 242 uses the key and the other contents of the notification to perform filtering and route lookup. Based on the filtering and route lookup, R unit 242 may modify the notification and forward the notification to memory system 240 or to RE 110. R unit 242 may also perform other types of processing. For example, R unit 242 might perform policing, such as L3 policing, sampling, multi-protocol label switching (MPLS), multicasting, and accounting support.

[0040] R unit 242 may include an input portion 310, a number of key engines 320, an external memory control 330, and a result cell processor (Rcp) 350. An external memory 340 may be connected to external memory control 330.

[0041] Input portion 310 processes keys and notifications from first I/O logic 236 and second I/O logic 238. Input portion 310 may include a buffer (not shown) for storing keys and

associated notifications. Input portion 310 may also include logic to distribute the received keys among key engines 320. In this manner, multiple keys may be simultaneously processed by key engines 320.

[0042] Key engines 320 may be connected to input portion 310, external memory control 330, and to Rcp 350. Key engines 320 may be configured to receive keys from input portion 310, and to perform route lookups for the keys in conjunction with external memory control 330 and external memory 340. Key engines 320 may store result data from the key processing in result buffers for transfer to Rcp 350. Key engines 320 may use internal memory (not shown) for storing results and other processing-related data. Such results may include, for example, one or more next hops for the packet of information associated with the processed key. In one implementation consistent with the principles of the invention, there may be 28 key engines 320 in R unit 242. Each key engine 320 may run multiple processes for processing keys. Key engines 320 will be described in greater detail with respect to Fig. 4 below.

[0043] External memory control 330 may be connected to key engines 320 and external memory 340. External memory control 330 may receive access requests for instructions from key engines 320. In one embodiment, access requests are received in a round-robin fashion. External memory control 330 may pipeline requests from key engines 320 to external memory 340 to fully utilize the bandwidth of external memory 340. External memory control 330 may also perform accounting, filtering, and policing functions for the key lookups.

[0044] External memory 340 may be connected to external memory control 330 and may be configured to store microcode instructions for processing the keys or other key-related information, such as forwarding tables and encapsulation tables. In one implementation consistent with principles of the invention, external memory 330 may include 16 megabytes of

double data rate synchronous random access memory (DDR SRAM). Such DDR SRAM may transfer data on both the rising and falling edges of an applied clock signal, effectively having a bandwidth of twice that of the clock signal. In one embodiment consistent with the invention, external memory may operate at 312 MHz, allowing R unit 242 to perform a route lookup for 80 million packets per second.

[0045] Rcp 350 may be connected to key engines 320. Rcp 350 may read result data from the result buffers for key engines 320, and modify the notifications from first I/O logic 236 and second I/O logic 238. In one embodiment, Rcp 350 services the result buffers for key engines 320 in a round-robin fashion. Rcp 350 may send the modified notifications to memory system 240 or to RE 110.

KEY ENGINE DESCRIPTION

[0046] Fig. 4 is a detailed block diagram illustrating portions of key engines 320 according to an implementation consistent with the principles of the invention. Each key engine 320 may perform table lookup, filtering, and route lookup. In one embodiment, use of external memory 340 and external memory control 330 are optimized by, for example, processing multiple keys within key engine 320. Internal memory (not shown) may also be used by the elements of Fig. 4. The number of keys concurrently processed by key engine 320 may be determined based on a ratio of a latency of memory 340 to an average time for processing a key. In the embodiment shown in Fig. 4, four keys may be concurrently processed using context switching.

[0047] Key engine 320 may include an input buffer 410, a data processor 420, a functional control state machine 430, a context buffer 440, a context switch controller 450, and an output buffer 460. Input buffer 410 may include a single segmented buffer or four separate buffers configured to store four keys and other data associated with four route lookup processes P0-P3.

[0048] Data processor 420 may be configured to process one key at a time using microcode instructions stored in memory 340. Data processor 420 may generate read addresses for memory 340 to access key-related information, such as forwarding tables and encapsulation tables, and use the information to compute parameters used in modifying the notification corresponding to the key being processed. During such processing (e.g., P0), data processor 420 may periodically read instructions or other data from memory 340 via external memory control 330. As will be described below, at that time, data processor 420 may be configured to request the data via output buffer 460, save any context state, such as partial results, in context buffer 440, and begin processing another key under control of context switch controller 450.

[0049] Functional control state machine 430 tracks an internal state of data processor 420 and provides such information to context switch controller 450. Functional control state machine 430 may be configured to inform context switch controller 450 that data processor 420 is about to request data from memory 340. Functional control state machine 430 also may be configured to store a state of the current process (e.g., P0) in context buffer 440 when data processor 420 requests data from memory 340.

[0050] Context buffer 440 may be configured to store context states, such as partial results, from data processor 420 and process states from functional state control machine 430 for four processes P0-P3. Because context states, such as partial results and process states, are stored in context buffer 440 during a data request for a process (e.g., P0), data processor 420 may continue processing another process (e.g., P1, P2) while P0 process would otherwise be idle. This storing of partial results and process states so that processing by data processor 420 may continue is called "context switching." Context switching effectively pipelines data requests from data processor 420, and avoids idle time for processor 420.

[0051] Context switch controller 450 may be configured to receive information from functional control state machine 430 that data processor 420 is about to request data from memory 340. In response to such information, context switch controller 450 may instruct data processor 420 to store a partial results and functional state control machine 430 to store a process states in context buffer 440. Context switch controller 450 also may be configured to load data processor 420 and context buffer with either a partial result and state from context buffer 440, or a new key from input buffer 410. In the first case, when data processor 420 resumes processing a stored process (i.e., a previously stored partial results), context switch controller 450 may also direct, for example, output buffer 460 to provide data returned from memory 340. Alternately, input buffer 410 may temporarily store the data returned from memory 340. Context switch controller 450 may include a first-in, first-out (FIFO) buffer (not shown) to determine what process (P0-P3) to load into data processor 420 and state machine 430 next.

[0052] Although the system of Fig. 4 has been described in terms of context switching while waiting for memory access request results, context switching may also be performed while additionally or alternatively waiting for other types of request results, such as processing request results.

[0053] Fig. 5 is an exemplary timing diagram 500 illustrating the context switching performed by key engine 320 according to an implementation consistent with the principles of the invention. The Clock signal may be common to and used by all elements in R unit 242, including key engine 320. The Process signal denotes which process (i.e., which key is being processed) is currently being performed by data processor 420. As illustrated in diagram 500, processes may perform one, two, or more calculations before needing data. In practice, different keys may be associated with different types of lookup processes.

[0054] The Calculation signal denotes various calculations performed by processes P0, P1, etc. The number of calculations performed by each process before needing data may vary in practice. For example, process P0 may perform three calculations before needing data or an instruction from external memory 340, while process P1 may perform only two calculations before a data or instruction request. In practice, the average number of calculations performed before needing data from memory 340 may be about three, but may range above or below this number. At time 510, data processor 420 and functional control state machine 430 respectively store a partial result and a state for process P0 in context buffer 440.

[0055] Data processor 420 may also make a request of memory 340 at this time. Memory 340 is one example of an "agent" from which data processor 420 may request information. Such requests are illustrated as the Agent Request signal. As may be seen in Fig. 5, multiple processes P0-P3 may make requests of multiple agents (e.g., Agents 1 and 2). Also at time 510, context switch controller 450 may cause a key and state for process P1 to be loaded from input buffer 410 into data processor 420 and functional control state machine 430. Processor 420 then performs calculations for process P1, as shown in diagram 500.

[0056] The above-described context-switching continues for processes P2 and P3. At time 520, process P3 may need data or instructions, and may make an Agent Request. By time 520, data D0 requested by data processor 420 for process P0 at time 510 may be available, as shown on the Agent 1 Data signal. At time 520, context switch controller 450 may cause a key and state for process P0 to be reloaded from context buffer 440 into data processor 420 and functional control state machine 430, along with data D0. Data processor 420 may resume performing calculation for process P0, performing one calculation for example, before again requesting data from an agent. When data processor 420 for process P0 again requests data, the

earlier-requested data D1 for process P1 may be available, and data processor 420 may perform, for example, three calculations for process P1.

[0057] When there are multiple agents, data may arrive faster from one agent than from another agent. For example, in Fig. 5, the second-requested D0 arrives from Agent 2 before the first-requested D3 arrives from Agent 1. Hence, process P0 may resume before process P3 (i.e., in a different order than the order in which they made the requests), because the data D0 for P0 arrives first and is available when process P2 halts at time 530.

[0058] Between times 520 and 530, processes P0-P2 may again be context-switched to and from context buffer 440. Such context-switching allows four keys to be concurrently processed by data processor 420, thereby more fully utilizing data processor 420 (see mostly utilized Calculation signal in Fig. 5). By pipelining data requests to external memory 340 and any other agents (see Agent Request signal), external memory 340 and any other agents are also more fully utilized.

[0059] It should be recognized that Fig. 5 is explanatory, and not limitative of the present invention. Details and timing conventions not explicitly discussed with respect to Fig. 5 will be apparent to those skilled in the pipeline processing art. For example, in one implementation the data requested by a process (e.g., D0 requested by P0) must arrive before processor 420 resumes that process. Also, the processor 420 may perform no calculations for one or more clock cycles if all processes are awaiting data or instructions from the memory 340 (see delay before second P2 processing in Fig. 5).

SYSTEM OPERATION

[0060] Figs. 6 and 7 are flowcharts of exemplary processing of a packet, according to an implementation consistent with principles of the invention. Processing may begin with a

network device 100 of Fig. 1, receiving a packet over a transmission medium, such as a WAN [act 610]. The packet may be one of several packets in a stream of packets transmitted between a source and a destination. Network device 100 may process the packet [act 620]. For example, network device 100 may strip the layer 1 (L1) protocol information from the packet.

[0061] Processing unit 232 or 234 may convert the packet into cells [act 630]. For example, the data of the packet may be divided into units of fixed size, such as 64 bytes, for storing in the cells. Processing unit 232 may also process the header of the packet, such as the layer 2 (L2) and layer 3 (L3) headers, and store L2 and L3 header information and the processing results in the cells. Further, processing unit 232 might create control information based on the packet. Processing unit 232 may also store the control information in the cells that it sends to first I/O logic 236.

[0062] First I/O Logic 236 may write the cells containing packet data into memory 240 [act 640]. First I/O logic 236 may store the cells in non-contiguous locations. Their location may be identified as a function of their relationship (offset) to the location of the previously stored cell in the memory 240. The address offsets may be stored in a notification [act 640]. If there are more address offsets than will fit in the notification, these additional offsets may be stored in an address cell memory.

[0063] R unit 242 may perform route lookup for the packet based on routing table(s) [act 650]. For example, R unit 242 may analyze the routing table(s) using information in the notification to identify a PIC from which the packet is to be transmitted. R unit 242 may store lookup information based on the route lookup in the notification [act 650]. The notification may then be forwarded to memory [act 650].

[0064] Returning to the system of Fig. 1, assume, for example, that the packet is received by a PIC connected to a first PFE 120 and is intended for a PIC of another PFE 120. In this case, second I/O logic 238 reads the cells and notification from memory system 240 and transmits them to switch fabric 130. Second I/O logic 238 may use data cell addresses 440 (Fig. 4) in the notification to read the cells from memory system 240. Switch fabric 130 transmits the cells and the notification to another PFE 120 (hereinafter "receiving PFE").

[0065] Fig. 7 illustrates a process of receiving cells from a switch fabric, such as switch fabric 130. The data cells are received from switch fabric 130 [act 710] (Fig. 7). The cells are written to memory. The cells may be stored in non-contiguous locations in the memory. The addresses of the cells as a function of their relationship (offset) to the memory location of the previously stored cell for the packet. The address offsets may be stored in the notification [act 720].

[0066] The cells are later read from the memory and transmitted [act 730]. The data cell addresses in the notification may be used to read the cells from the memory. Updated notification information may be stored in the cells.

[0067] A packet may then be constructed from the cells and the notification [act 740]. For example, in the system illustrated in Fig. 2, processing unit 234 may extract the notification, control information, and packet data from the cells and create a packet therefrom. Processing unit 234 may construct a packet header, such as L2 and/or L3 headers, from the notification and control information and load the packet data portion with the packet data in the cells.

[0068] The packet may then be transmitted on a transmission medium, such as a WAN [act 750]. The packet may also be encapsulated in L1 protocol information before sending the packet out on the WAN.

R UNIT OPERATION

[0069] Fig. 8 is a flow chart illustrating processing performed by R unit 242 according to an implementation consistent with the principles of the invention. Processing may begin with input portion 310 receiving a notification from first I/O unit 236 or second I/O unit 238 and sending a key from the notification to one of key engines 320 [act 810]. Key engine 320 performs route or encapsulation lookup based on the key [act 820]. In conjunction with the route lookup, external memory control 330 may perform accounting, filtering, and policing operations based on the key and the lookup operation [act 830]. Using the results of these operations, result cell processor 350 may modify the notification [act 840] and forward the modified notification to memory 240 or RE 110 [act 850]. Although the acts of Fig. 8 are illustrated sequentially, non-dependent acts can be performed in parallel and in a different order. Additionally, other acts described in reference to Figs. 6 and 7 may also be performed in parallel to the acts described in reference to Fig. 8 and in a different order where dependencies between the acts allow.

KEY ENGINE OPERATION

[0070] Fig. 9 is a flow chart illustrating processing performed by key engine 320 according to an implementation consistent with the principles of the invention. Processing may begin with data processor 420 receiving a key to process and with state machine 430 receiving an initial state from input buffer 410 [act 910]. Data processor 420 may process the key until it is either finished processing or needs data or an instruction from memory 340 [act 920].

[0071] If data processor 420 finishes processing the key [act 930], it may store the result associated with key in output buffer 460 [act 940]. If, however, data processor 420 is not finished processing the key (i.e., it needs data or an instruction from memory 340) [act 930], data processor 420 may request such data or instructions from memory 340 [act 950].

[0072] At this time, under control of context switch controller 450, the current key may be context switched and its partial result and processing state may be stored in context buffer 440 [act 960]. When either the result is stored in output buffer [act 940] or the current key is context switched to context buffer 440 [act 960], context switch controller 450 may determine which key is to be processed next [act 970]. Context switch controller 450 may use a FIFO buffer to make such a determination. If data has been returned from memory 340 and an existing key's process is to be resumed, data processor 420 may load a stored partial result and state machine 430 may load a stored state from context buffer 440 [act 980]. Processing of the existing key may continue as shown in acts 920, 930, etc.

[0073] However, if context switch controller 450 determines that processor 420 should start with a new key, data processor 420 may receive the new key and state machine 430 may receive an initial state from input buffer 410 [act 910]. Processing of the new key may continue as shown in acts 920, 930, etc. In this manner, key engine 320 may process several keys concurrently, thereby keeping data processor 420 busy through the use of context switching.

[0074] Although described in the context of a purpose-built router, concepts consistent with the principles of the invention can be implemented in any system that requires high performance data item processing. Apparatus, systems, and methods based on the principles of the routing unit or key engines described herein may be used in any environment for processing data items associated with an entity. The data items are processed using context switching for the entities. Entities may include sources of data items, as described herein, or other entities, such as destinations, processing threads, or any other entity having individual data items that must be processed.

[0075] The foregoing description of preferred embodiments of the invention provides illustration and description, but is not intended to be exhaustive or to limit the invention to the precise form disclosed. Modifications and variations are possible in light of the above teachings or may be acquired from practice of the invention.

[0076] For example, although the present invention has been described with four concurrent processes per key engine, fewer or more keys may be processed per key engine. For example, from two to ten or more keys may be concurrently processed by a single key engine using context switching in accordance with the principles of the invention. The number of concurrent key processes per key engine may depend on a ratio of memory latency time to an average processing time between memory access requests.

[0077] No element, act, or instruction used in the description of the present application should be construed as critical or essential to the invention unless explicitly described as such. Also, as used herein, the article "a" is intended to include one or more items. Where only one item is intended, the term "one" or similar language is used. The scope of the invention is defined by the claims and their equivalents.